Carnegie Mellon
Software Engineering Institute

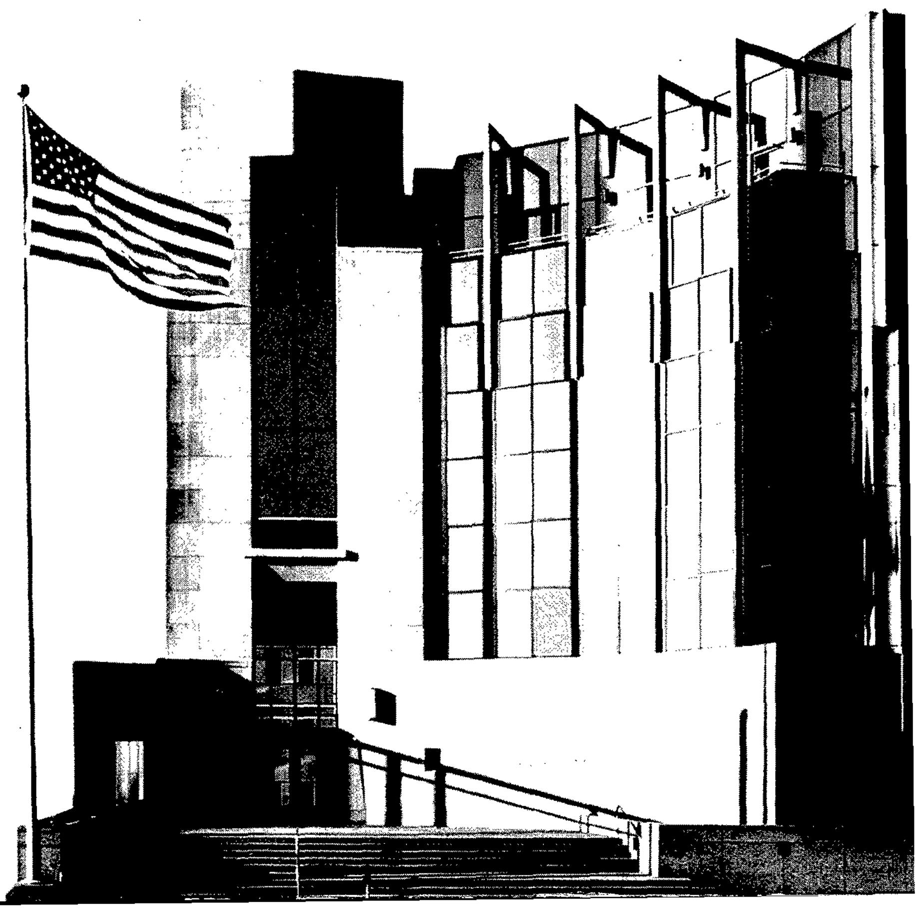
# Model Problems in Technologies for Interoperability: Model-Driven Architecture

Grace A. Lewis
Lutz Wrage

*May 2005*

**Integration of Software-Intensive Systems (ISIS) Initiative**





**Technical Note**
CMU/SEI-2005-TN-022

# Model Problems in Technologies for Interoperability: Model-Driven Architecture

Grace A. Lewis
Lutz Wrage

*May 2005*

**Integration of Software-Intensive Systems (ISIS) Initiative**



**Technical Note**
CMU/SEI-2005-TN-022

This work is sponsored by the U.S. Department of Defense.

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.



For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

# Contents

# List of Figures

# List of Tables

# Abstract

Model-driven architecture (MDA) is a technology produced and maintained by the Object Management Group (OMG), an open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications. This technical note examines two claims regarding the benefits of MDA, namely, that it (1) reduces development time, and (2) allows the developer to focus on business logic rather than on details about the target platform and architecture. Such advantages would greatly benefit interoperability; as target platforms and underlying infrastructure change, deployment of applications would be quick and easy. This note presents the results of applying the model problem approach to verify these claims.

# 1 Introduction

From a technology perspective, there are many current approaches to building systems with interoperability requirements. Each has particular advantages and disadvantages with respect to interoperability, and each works well in some circumstances but not others [Lewis 05]. This report will look at Model-Driven Architecture (MDA) as one of many technologies for accomplishing interoperability.

Model-Driven Architecture (MDA) is an approach to software development produced and maintained by the Object Management Group (OMG), an open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications. The goal of MDA is one that is often sought: to separate business and application logic from its underlying execution platform technology so that (1) changes in the underlying platform do not affect existing applications, and (2) business logic can evolve independently from the underlying technology [Lewis 05, OMG 03]. A tool that implements the MDA concept allows developers to (1) produce models of the application and business logic, and (2) generate code for a target platform by means of transformations.

MDA is not to be confused with Model-Driven Development (MDD), also known as Model-Driven Software Development (MDSD). MDD is an approach to software development where extensive models are created before source code is written or generated. MDA is the OMG implementation of MDD and it is built upon existing OMG standards, such as UML. The MDA concept is implemented by a set of tools and standards that can be used within an MDD approach to software development. Even though MDD is often associated with agile software development processes, it can also be used within other processes where modeling plays a big role, such as Object-Oriented Analysis and Design (OOAD) and the IBM Rational Unified Process (RUP).

The Software Engineering Institute is examining technologies and approaches for the construction of systems that are required to interoperate with other systems, with the purpose of examining the gaps between what these technologies and approaches offer and what users expect of them. The end goal is to provide users with information about what can be expected by the current state of technology and to provide technology suppliers with information about user expectations, hoping to help bridge these gaps.

MDA literature, and some MDA tool vendors, claim that the MDA approach (1) reduces development time, and (2) frees the developer to focus on business logic rather than on target platform and infrastructure details. Aside from being a valid and beneficial approach to software development, this should allow quick and easy deployment of applications, as target platforms and underlying infrastructure change. If realized, these advantages would significantly benefit interoperability.

To verify these claims about MDA, we chose the model problem approach [Wallnau 01]. This involves (1) formulating hypotheses about the technology, and (2) examining these hypotheses against very specific criteria through experimentation. In this way the hypotheses are either sustained or refuted. The model problem approach has the advantage of producing very efficient and representative experiments that not only evaluate technologies within the context of their future use, but also generate hands-on competence with the technologies.

Section 2 explains the model problem process. The details of the experiments are presented in Section 3. Results are presented in Section 4. Section 5 provides details about the experience with MDA. And finally, Section 6 presents some conclusions along with recommendations for users and vendors of MDA-based development tools.

# 2 Model Problem Approach

Model problems were initially created as a technique for evaluating commercial components [Wallnau 01]. We have slightly modified the original model problem process so it extends to the evaluation of technology in general. A graphical representation of the model problem process is presented in Figure 1. The model problem is part of a larger process for context-based technology evaluation.[1] In this larger process, the context for the model problems is established and the expectations from the technology are captured.

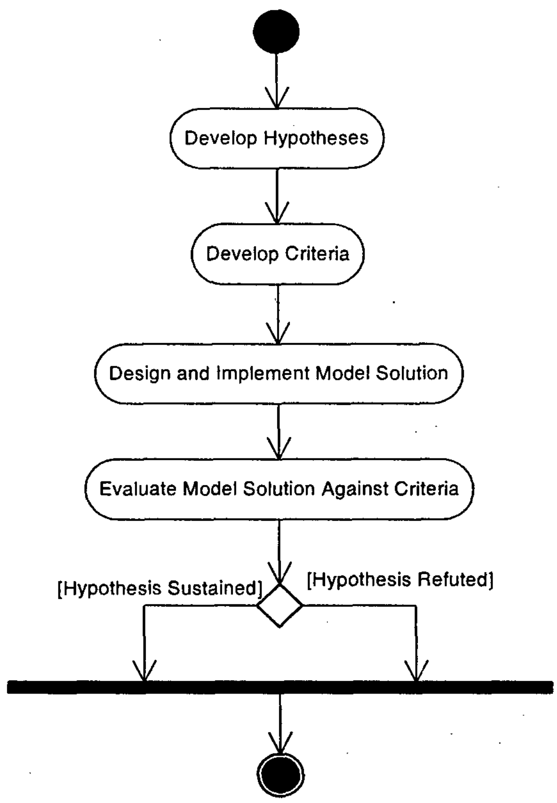



*Figure 1: Model Problem Process for Technology Evaluation*

The context for our model problem is a military human resources system we worked with recently. Military personnel are often reassigned to new locations, which triggers additional processes such as the actual reassignment, payroll adjustments, and flight booking. We were also specifically interested in the Java 2 Enterprise Edition (J2EE) platform because component frameworks are another technology of interest for interoperability.

1 Lewis, Grace A. & Wrage, Lutz. *Context-Based Technology Evaluation*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University. To be published.

## 2.1 Develop Hypotheses

The first step in the process is to define hypotheses. Hypotheses are claims about the technologies that are to be sustained or refuted. For MDA we defined the following initial hypotheses:

1. The use of MDA reduces development time.
2. The use of MDA-based development tools frees the developer from learning the low-level target platform and underlying infrastructure details.

## 2.2 Develop Criteria

Criteria are used to determine if a hypothesis is sustained or refuted. Each hypothesis can have one or more criteria, depending on the breadth covered by the hypothesis. Each criterion is stated as a clearly measurable statement of capability. For the above hypotheses, the defined criteria are described in Table 1.

| Hypothesis | Criteria |
|---|---|
| The use of MDA reduces development time. | The time to develop a J2EE application using an MDA-based development tool will be less than it would take us to write the application using a standard IDE. Times will be compared against data from previous developer experience. |
| The use of MDA-based development tools frees the developer from learning the low-level details of the target platform and underlying infrastructure. | The developer will need to understand J2EE at a conceptual level but will not need to have a detailed understanding of J2EE or the JBoss application server while generating a J2EE application for JBoss using an MDA-based development tool. |

*Table 1: Hypotheses for MDA*

## 2.3 Design and Implement Model Solution

A model solution is the mechanism by which data is collected to sustain or refute a hypothesis or set of hypotheses. Building a model solution for technology evaluation in this context has three parts.

1. Define scenario.
2. Define technical solution that satisfies the scenario.
3. Implement technical solution.

It is important to ensure that all hypotheses can be sustained or refuted with the defined set of scenarios. Testing more than one hypothesis using the same scenario may be possible if it is carefully designed.

The model solution should be very efficient—it only needs to answer the question at hand. The purpose of the scenarios is to ensure that the model solution is realistic and implemented within the context of how the technology will be used. Therefore, a model solution should involve the simplest set of applications and/or components that are able to sustain of refute the hypotheses as measured by the associated criteria, given the context framed by the scenario.

Hypotheses: see above

Scenario: A military organization has a Human Resources (HR) system that maintains personnel records. Some of the information maintained for each person is: name, complete address, and assignment location. When the personnel record is created, the person is given his/her initial assignment location.

Technical Solution: An MDA-based development tool is used to create the HR system as a Java 2 Enterprise Edition (J2EE) application that uses a number of Enterprise Java Beans (EJBs) to perform basic Create-Retrieve-Update-Delete (CRUD) operations and basic validations on personnel data. The user interface is implemented as a set of Java Server Pages (JSPs) that are accessed through a browser [Sun 05]. Apache Jakarta Tomcat is used as the servlet container [Apache 05]. The J2EE application server used is JBoss Application Server [JBoss 05]. Data is stored in an Oracle database [Oracle 05].

We present the details of implementing the technical solution in Section 3.

## 2.4 Evaluate Model Solution Against Criteria

In this step, the technical solution is implemented, run, and observed, until there is enough information to sustain or refute the set of hypotheses. Criteria sometimes require refining because they are too vague, making it difficult to decide whether the hypothesis is sustained or refuted. Sometimes new criteria must be created based on findings during implementation.

We kept notes about product installation, problems encountered, interim results, deployment environment, and any other information that might help future projects, should the technology be selected for use in the organization. For this set of model problems we used a blog that was shared with all members of the team.

Section 4 contains details of the evaluation.

# 3 Implementing the Technical Solution

The first step in implementing the technical solution was finding an MDA-based development tool. Given that we were interested in evaluating the technology and not the tool itself, we did not want to perform an extensive evaluation and selection process. The initial selection criterion was very simple: the tool had to be able to generate code for both J2EE and .NET, because we did not want to learn a separate tool for other future model problems involving .NET. If there were many tools that did this, we would add additional criteria to reduce the size of the pool. As a source for tools, we relied on the success stories published on the OMG site.[2] We made a list of all the tools mentioned in these stories and began visiting their Web sites to see which ones would satisfy our initial criterion. To our surprise, only two tools did. We sent email to both tool companies, requesting an evaluation license and explaining what we were going to do with the tool. Only one of the companies, Interactive Objects, answered our request; shortly after they provided a license for its tool ArcStyler [IO 05].

In parallel, we began designing the architecture for the system. Figure 2 shows a high-level component and connector view for the HR System as a typical J2EE application. Figure 3 shows a deployment view.

[2] http://www.omg.org/mda/products_success.htm

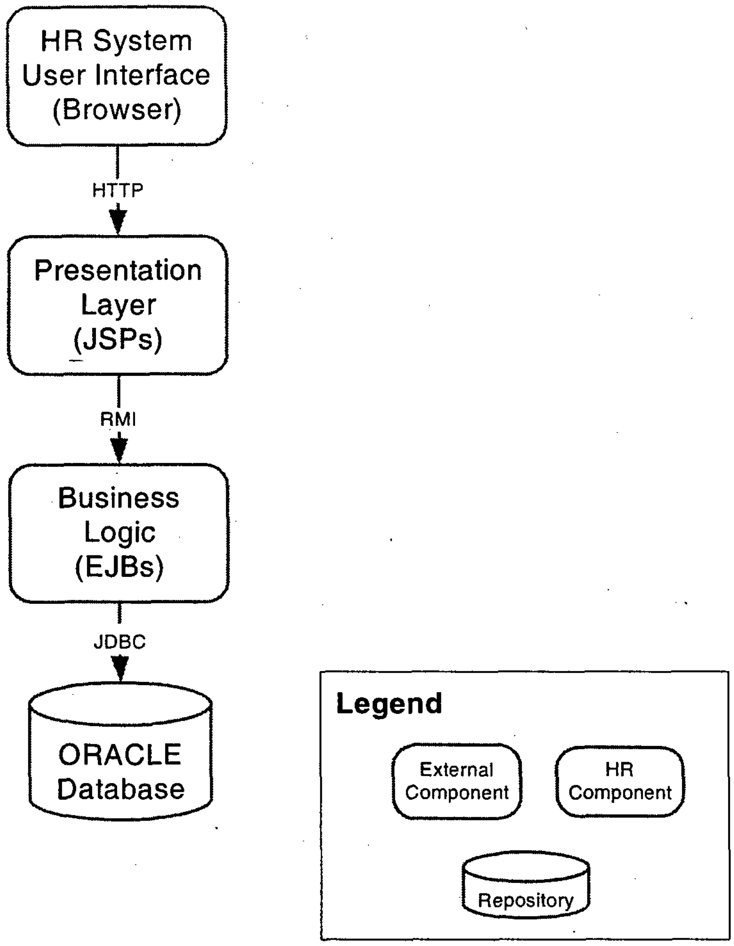



*Figure 2: High-Level Component and Connector View for the HR System*

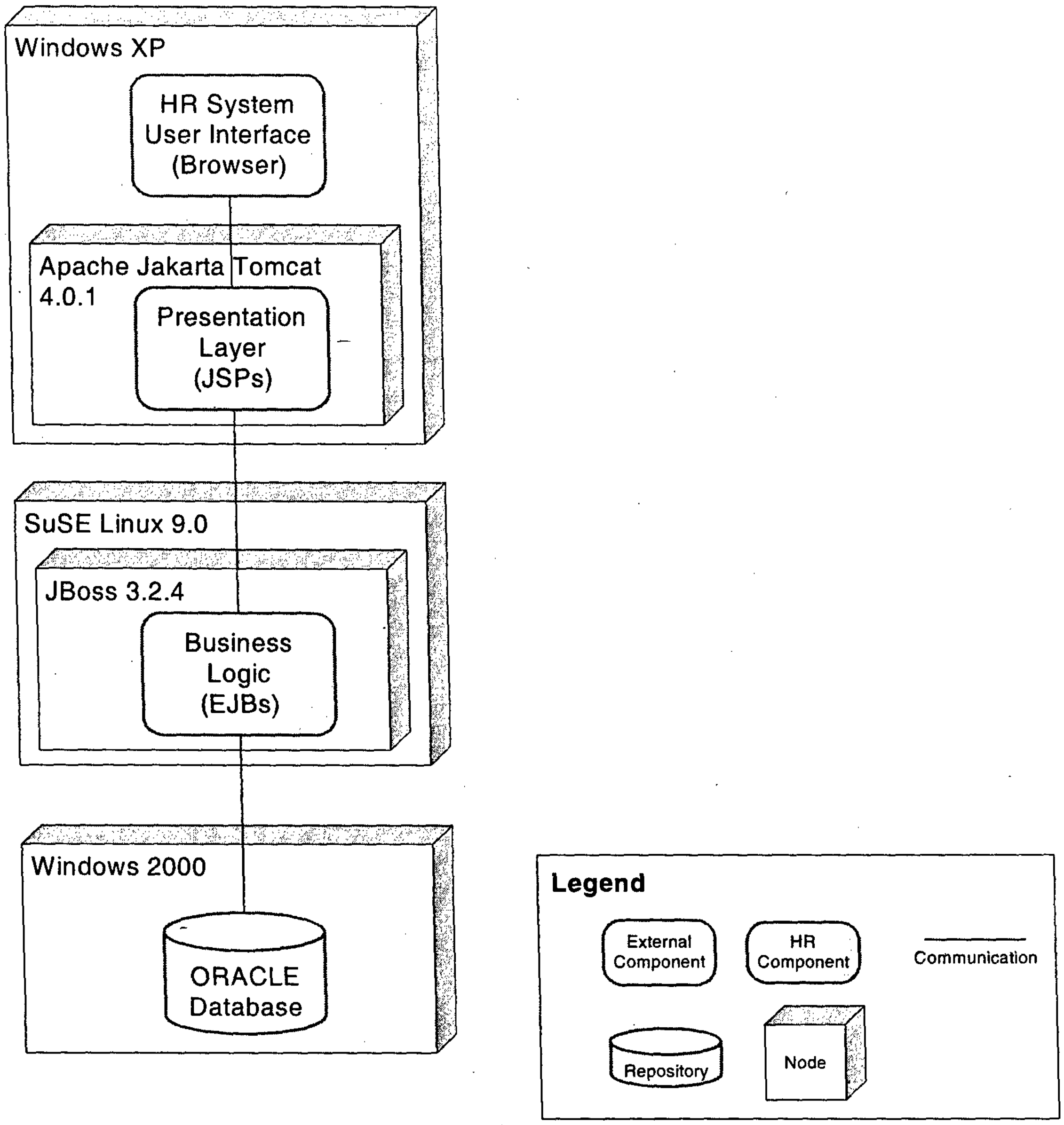



*Figure 3: Deployment View for the HR System*

The ArcStyler tool uses the Unified Modeling Language (UML) as the modeling notation [OMG 05c]. The tool elements that transform the models into code are called cartridges.

We used the ArcStyler tool to generate code for the following components:

- EJBs for the business logic and data management elements of the HR system for the JBoss application server using the JBoss32 cartridge
- JSPs for the user interface of the HR system using the WebAccessors cartridge

The evaluation of the model solution against the model problem criteria is presented in the next section.

# 4 Evaluation

The last step in the model problem process is to evaluate the model solution against the criteria in order to determine whether the hypotheses are sustained or refuted. The results of this process are provided in the following sections.

## 4.1 Hypothesis 1: The Use of MDA Reduces Development Time

This hypothesis is partially refuted. Development time is not reduced for the development of the first application for which a tool is used, but it may be considerably reduced for subsequent applications developed for the **identical** platform. When the platform is not identical, tools require reconfiguration and transformations[3] may require modification. To confirm this, we used the tool a second time to generate the JSPs for the user interface of another model problem that we were working on. The development time was significantly reduced for the second application we generated with the tool mainly because it was targeted at the identical platform: JSPs running in a Tomcat 4.0.1 servlet container connecting to a JBoss 3.2.4 application server. Making this "model-to-code" process repeatable so that development time is reduced requires discipline in describing the process.

Several studies about MDA and productivity have been conducted, such as the one produced by The Middleware Company in 2003 [Middleware 03]. This study claims a 35% reduction in development time was realized by using MDA. Even though we believe these results, we also believe that the tool was used mostly as-is, did not count training time, and the process did not involve the configuration and potential transformation modifications that would most probably be required in a real-world situation. We believe that all these elements require large amounts of time and should not be underestimated.

A more detailed description of our findings follows.

### 4.1.1 Difficult Learning Curve

The learning curve for MDA-based development tools is very steep. While this is somewhat true for any new development tool, we make our assessment based on extensive experience with such tools. Both participants in the model problem process have approximately 15 years of development experience and one participant has previous experience with two other MDA-based development tools. Both found the learning curve for MDA to be much steeper than for other development tools.

[3] Transformations are the set of rules that convert a platform-independent model to a platform-specific model. This concept is further explained in Section 4.1.3.

The difficult learning curve is due to some of the components in an MDA-based development tool. We list them all to provide an idea of the complexity of some of these tools:

- Modeling component – In the tools we have worked with, the modeling component uses UML notation. UML is fairly well known. The user interface varies from tool to tool but it is not difficult to learn so it does not contribute to the difficult learning curve.
- Platform-specific model configuration component – The platform-specific model configuration component is tool specific. In the case of ArcStyler, model configuration is done through parameter settings in tabs for each of the components of the model, as shown in Figure 4 and Figure 5. Each component requires configuration so that the code generated for it matches the requirements of the technical solution.
- Transformation component – The transformation component is called cartridge, model transformer, code generator, or translator, depending on the tool. Invoking the transformation component is fairly simple; what is time-consuming is understanding what it generates and where it generates it. Given that most MDA-based development tools are limited to generating code from UML models, transformations are commonly referred to as code generators. This will be addressed further in the findings for the next hypothesis.
- Build and deployment component (optional) – The build and deployment (or simulation) components are optional. These are also fairly straightforward to invoke and use and do not contribute greatly to the difficult learning curve.
- Transformation editor component – The transformation editor is probably the most difficult of the components to use. Transformations need to be modified when there are mismatches with the target platform. This will be addressed shortly.

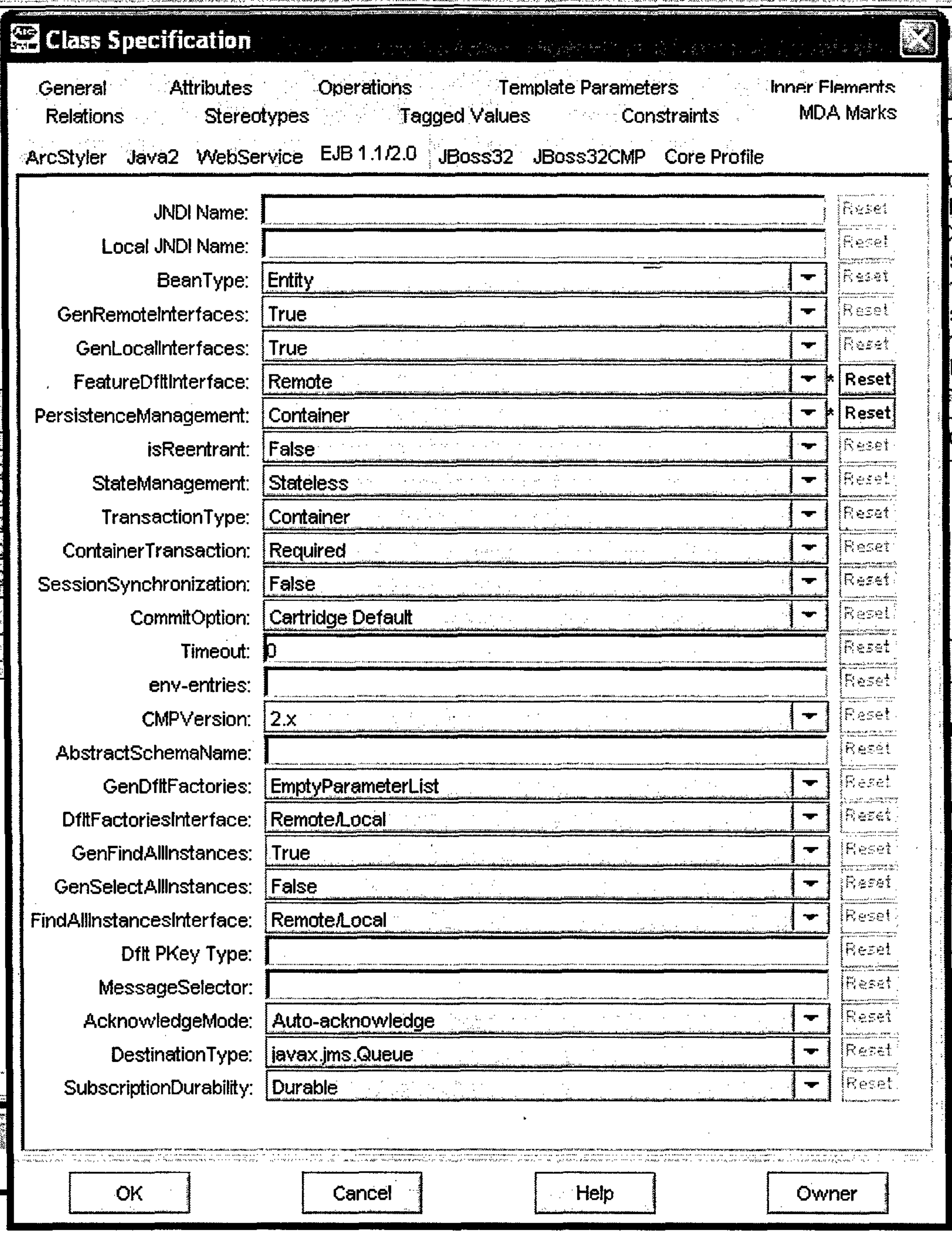



*Figure 4: Component Configuration Screenshot—EJB Configuration*

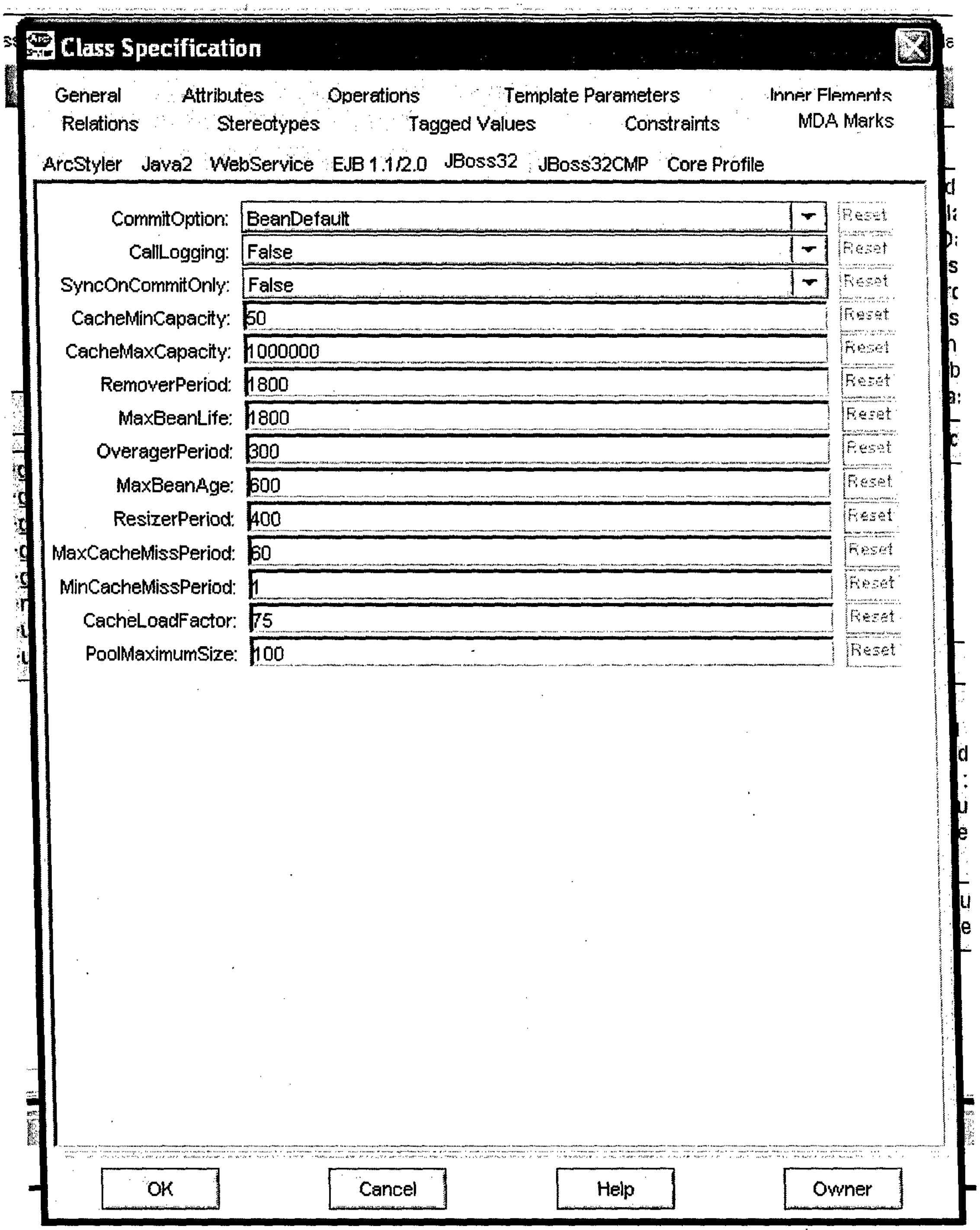



*Figure 5: Component Configuration Screenshot—JBoss Configuration*

### 4.1.2 Configuration and Debugging Is "Trial-and-Error"

During configuration there is a lot of trial and error—looking at the generated code to analyze any problems, changing configuration parameters in the tool, and repeating this process until the application works. Fortunately, repeating this process can be avoided if all configuration

settings are documented. This contributes to reduced development time for subsequent applications for the identical platform and is not unique to MDA-based development tools.

### 4.1.3 Transformations Generate Code for a Very Specific Platform

Changes to transformations can be a time-consuming and difficult task. In general, in tools that implement the MDA concept, a platform-independent model (PIM) that contains only business and application logic is developed using the tool's modeling component. The code generation component is a series of transformation rules that map elements in the PIM to elements in a platform-specific model (PSM) that contains details that are specific to the target platform. Code is considered a form of PSM and a PIM can go through several levels of transformation before becoming code. A simple example of a set of high-level transformation rules for the J2EE platform, not specific to any tool, is shown in Figure 6. This model transformation process is illustrated in Figure 7.

```
if (UMLClass) {
    create Java class named <UMLClass.className>Bean.java
    create methods in <UMLClass.className>Bean.java for each
      operation in UMLClass
    create attributes in <UMLClass.className>Bean.java for
      each attribute in UMLClass
    create Java class named <UMLClass.className>.java for
      remote component interface
    create Java class named <UMLClass.className>Home.java for
      remote home interface
    create Java class named <UMLClass.className>Local.java for
      local component interface
    create Java class named <UMLClass.className>LocalHome.java
      for local home interface
    ...
}
```

*Figure 6: Sample Transformation Rule*

As Figure 6 shows, transformation rules are very platform specific. Many things that one might wish to change in a transformation are not configurable via settings, such as file location, personal preferences, coding standards, and organizational user interface standards. In more extreme cases mismatches occur between the transformation and the target environment. For example, if the tool provides a C code transformation for a stand-alone environment, the transformation would have to be largely modified or rewritten completely if the target is C language for a distributed environment with real-time concerns.

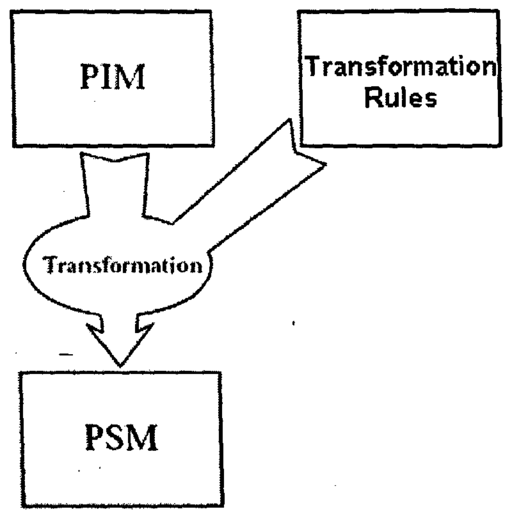



*Figure 7: MDA Model Transformation Process*

Modifying these transformations is not a trivial task and may require learning a completely new scripting language. ArcStyler, for example, uses a language called Jython [Jython 05]. The MDA-based development tools that we previously worked with offered full courses and manuals on how to tailor the transformations, or write your own, acknowledging that these are not easy tasks.

To conclude, if the transformations need to be written entirely or heavily tailored, the development time will not be reduced, and might even be increased, for the first application for which MDA is used, with eventual, potentially large savings for subsequent applications.

### 4.1.4 Not All Tools Generate the Same Amount of Code

One element that factors into development time is the amount of code that must be written that is not generated by the tool. There is currently a wide variation in the amount of code that an MDA-based development tool generates. There are tools that generate infrastructure code but do not generate business- and application-logic code. Typical examples are tools that generate code for J2EE and .NET environments. In these cases, the amount of code still to be written can be large, depending on the complexity and size of the system. Conversely, there are tools that implement what is called Executable UML (xUML). In the xUML approach, business logic is expressed in the models as state machines and an accompanying action language. In this case, the application and business logic is generated by the tool as well, minimizing the amount of code to be written.

As attractive as the idea of not having to write code can be, we have learned from experience with other MDA-based development tools that representing all business and application logic in a system as a state machine can be difficult and not intuitive for all types of systems. For example, representing a bank account as a state machine would include states such as open, closed, suspended, and overdrawn, and there are clear rules that take an account from one state to another. However, there are many static-like entities in a model that usually

correspond to lookup data, that do not change and for which it does not make a lot of sense to create a state machine. Also, the action language provided by these xUML-based tools is not complete. For example, if the business logic requires complex mathematical operations, these would have to be provided by external code.

In short, depending on the tool used, there is still work involved in writing business- and application-logic code. ArcStyler, for example, indicates by using the tag @TODO within the code that there is logic code to write corresponding to the body of the methods in a class. All user-written code is included within protected areas so that it is not deleted when the application is re-generated. This approach requires developers to understand the logic of the generated code so that they can add the appropriate code at the right places.

## 4.2 Hypothesis 2: The Use of MDA-Based Development Tools Frees the Developer from Learning the Low-Level Details of the Target Platform and Underlying Infrastructure

This hypothesis is refuted. Assuming that development starts from the moment a design is delivered or finalized, the tool, models, and transformations must be configured before any type of code is generated. This configuration requires very good knowledge of the infrastructure target platform. The use of MDA-based development tools frees the developer from having to write infrastructure code, but it does not free the developer (or whoever is doing configuration) from learning the low-level details of the target platform and underlying infrastructure. A more detailed description of our findings follows.

### 4.2.1 Three Different Roles: Designer, Developer, and Configuration Expert

MDA introduces a third role in the development process: the configuration expert. It is not uncommon in software development to separate the role of designer and developer. The designer is a modeling expert who produces a model of the system. The developer takes the model and writes code that implements it. The designer might be a UML expert and the developer might be a Java/J2EE expert. When using an MDA-based development tool, there is a step in between modeling and producing code: configuration. In our discussion of the previous hypothesis, we saw that during configuration, certain global and component-specific parameters are set so that transformation is done properly. While it is true that all infrastructure code is generated and therefore the developer need not worry about this task, the configuration itself and the trial-and-error process mentioned above require low-level detailed knowledge of both the target platform and the underlying infrastructure.

There are many configuration parameters that require platform knowledge. In our model problem, one of our tasks was to generate EJBs for the business- and data-management-logic of the HR system, to be deployed on a JBoss application server. For example, for each class in the UML model corresponding to our system (see Figure 8), we had to define it as a session or entity bean, add *create* methods, and identify which methods to include in the local

and remote interfaces. Knowledge of J2EE is required to determine the appropriate configuration. In many cases, when an application did not work properly, we had to look at the generated code and determine the potential problem, change some parameters in the tool, and generate again until the application worked. This required understanding the code, as well as knowledge of the appropriate configuration outcome. Also, transformations sometimes come with built-in assumptions. For example, the JBoss32 cartridge that we used has configuration parameters for a Hypersonic database. However, to use a different database, such as Oracle, the configuration must be done manually (this changed in the current version of ArcStyler). Setting a data store requires knowing what files to modify and to deploy, the details of which are highly dependent on the J2EE application server used.

User interface transformations will probably require defining a template for the screens if the default template is not appropriate for the organization's needs and standards. For example, the template may require inclusion of the corporate logo, window styles, menu styles, and all screen standards. For the ArcStyler WebAccessor cartridge, templates are defined using JSP and are then included in the generated JSP files corresponding to the user interface. For the model problem we made only very small modifications to the templates.

In short, the configuration expert will require both MDA development tool and target environment knowledge because it is required especially when using the tool for the first time. This role should be assigned to the designer, a developer, or a third person. For subsequent applications this knowledge requirement might be reduced if configuration steps are provided as a "recipe" or are encoded in the transformation. However, at least conceptual knowledge would be required to make decisions and ideally a target platform expert should be on the development team.

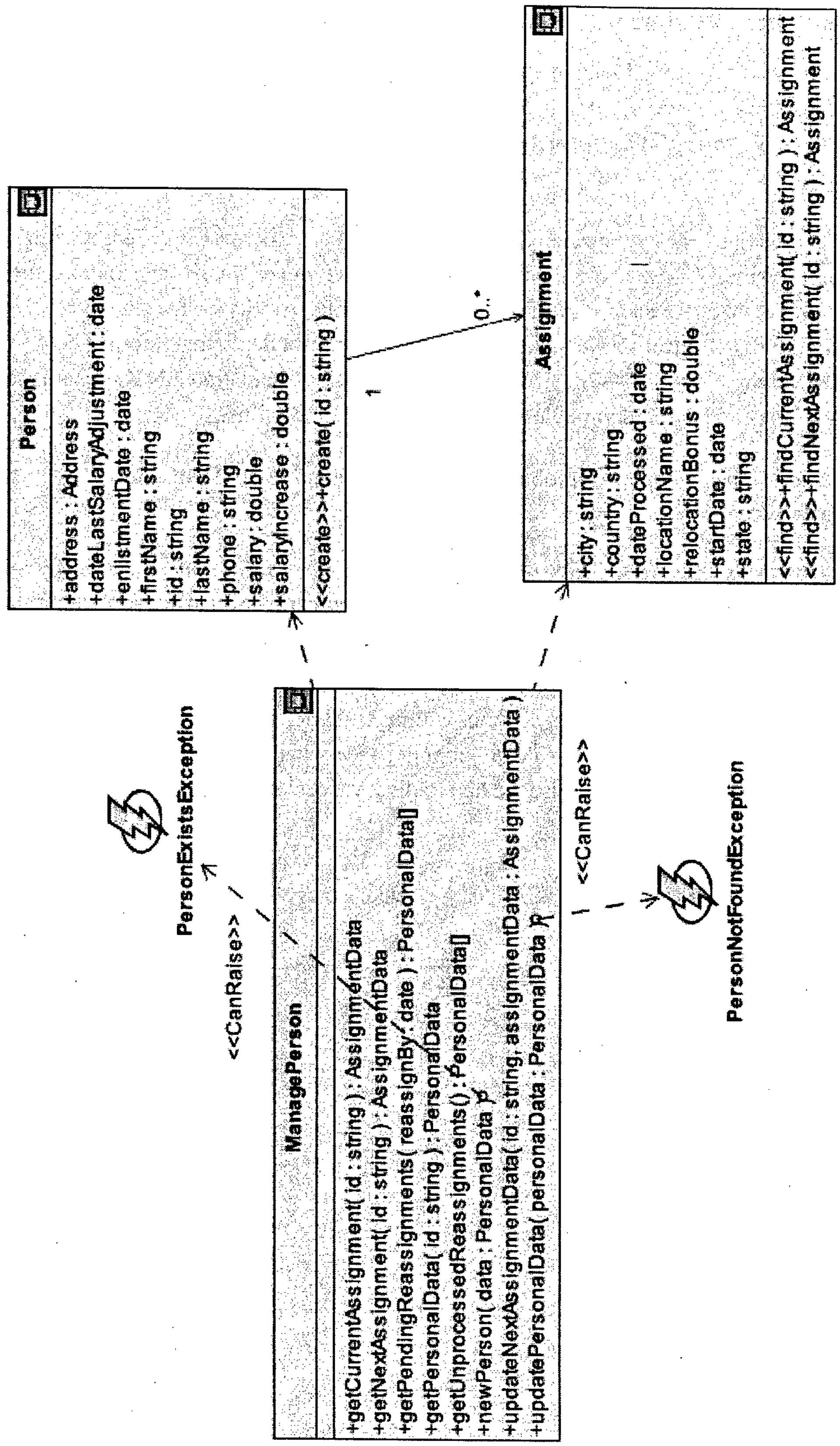



*Figure 8: Class Diagram for the Business and Data Management Logic of the HR System*

### 4.2.2 Deployment Requires Platform and Infrastructure Knowledge

It is probably true that any development environment requires platform and infrastructure knowledge. However, since MDA-based development tools commonly advertise that it is not necessary to have platform and infrastructure knowledge, it is important to mention this point.

Deployment can be quite tricky, especially if the deployment environment is different from the deployment or simulation environment included with the tool so that testing can be done on the development machine instead of the production machine. ArcStyler, for example, comes with an internal Tomcat servlet container as well as the Ant deployment tool. The MDA-based development tools we previously used came with a simulation environment that showed the values of objects and variables at simulation execution time.

The following are some examples of problems we had that required platform and infrastructure knowledge.

- We had several problems with Tomcat because the version within ArcStyler was different from the one that was installed in the deployment environment. Even after we thought the application was working, we had to manually change several configuration and deployment files so the application would actually work.
- We had to switch from running Tomcat from within ArcStyler to running it from the Eclipse IDE so we could do server-side debugging to see what was happening on the JBoss side. This was difficult because we had to discover the exact Tomcat settings that ArcStyler used.

# 5 Experience with MDA

Our exploration of the MDA concept through the use of a model problem has been quite interesting. We have gained competence in several technologies while examining various claims about MDA and the circumstances under which they prove true or false. The vendors of the technologies used in the model problem have been very helpful providing software, documentation, and support. Exchanges with the ArcStyler vendor about future plans for the company's tool support some of our findings and recommendations about the future direction for MDA-based development tools in general. As expressed earlier, our interest was in the evaluation of the technology and not the tool itself. Therefore, none of the following points should be interpreted as pertaining specifically to ArcStyler. We are also using the experience with two other MDA-based development tools, as well as data from the tool selection process discussed earlier.

## 5.1 Development Time is not Reduced for the First Application for which an MDA-Based Development Tool is Used

While MDA-based development tools work well for modeling and code generation, they involve a large initial investment in configuration and potential transformation modifications. This leads us to conclude that if the tool is going to be used for just one application, or one application on multiple platforms, its use should be reconsidered, unless expert help is hired or the tool is a perfect match for the target platform.

In our experience, development time does decrease for subsequent applications developed for the identical platform. A large initial investment in configuration and transformation modification may be reasonable in industry where there is usually a common deployment platform, but in the military where the same application is commonly deployed on multiple platforms, this might be a problem.

## 5.2 The Real Potential of MDA is not Completely Supported by Current Tools

The MDA vision is very much dependent on the availability of tools that enable developers and organizations to realize the promised benefits. In our experience, current MDA-based development tools implement only part of the MDA concept. Current tools are able to generate code from models, which is one aspect of the MDA concept. But these tools fail to take advantage of another important aspect of MDA: the use of common modeling and transformation standards that allow models to be successfully shared between tools without information loss. To provide a common modeling and transformation representation for

vendors to use in their tools and therefore allow sharing of UML models, OMG is working on specifications such as XML Metadata Interchange (XMI); Meta-Object Facility (MOF); Common Warehouse Meta-model (CWM); and Query/Views/Transformations (QVT) [OMG 02a, OMG 02b, OMG 05a, OMG 05b]. QVT is still in proposal stage and has not yet become a standard.

Most tools we surveyed and used are restricted to one level of transformation: from a UML model directly to code. This limitation requires that all information for the code generation must be included as transformation-specific markings in the PIM. In particular, if there are several target platforms for the application, then all markings for all target platforms must be part of one model. Developing and maintaining this model in a team may become problematic from a configuration management perspective.

Most tools are also equipped to generate code for only one target platform. When selecting a tool to implement our model problem, we conducted a short survey of 10+ tools, as indicated in Section 3. Out of these, only two tools were capable of generating code for our two target platforms. Most other tools were bound to one platform, mainly J2EE. It is our impression that some MDA-based development tools are little more than code generators for J2EE applications.

If tools could successfully exchange PIMs, developers could then use different tools to generate code for different platforms while still using the same underlying model. The OMG has addressed model exchange with the XMI standard. In theory, tools that store models in XMI format should be able to exchange models. In practice, however, some loss of data is likely to occur because of differing interpretations of the XMI standard. There is no guarantee that an XMI document produced by one tool can be consumed without loss of data by another tool. An additional issue for model exchange is the detail that is necessary in the models for a tool to work. Most prominently, tools that support xUML require the model to contain all information necessary to generate the complete source code; this is unnecessary for the tools that generate only infrastructure code and leave the implementation of business logic to the developer [Mellor 02]. These are two very different paradigms which add to the difficulty of model interoperability.

Exchanging models is just one building block of MDA-based tool interoperability. The next level of interoperability requires tools that can exchange markings and transformations. This requires a new generation of tools that are not available today—not surprising since the standard representation of transformations does not yet exist. Currently, transformation definitions are tool specific. As we mentioned earlier, this means that a developer who needs to tailor the transformation must learn from scratch how to accomplish this. QVT, currently under development, is intended to address this issue.

All tools we experimented with use UML as the modeling language. While UML is widely used and extensible via the stereotype mechanism, it may not always be the ideal choice for modeling. Petri-nets, for example, would be useful to reason about synchronization of

concurrent processes and these cannot be used with any of these tools.[4] The MDA concept assumes that developers can define their own, domain-specific modeling languages using the OMG's meta-object facility (MOF). Support for this type of tool extensibility is not yet widely available.

An ideal tool could represent models in a common abstract component model style corresponding to the PIM. An initial transformation could be applied to the model to create a J2EE-specific model (PSM). A second transformation could take the J2EE model to a JBoss- or WebLogic-specific platform and therefore code (lower level PSM). The same PIM could be transferred to a different tool so that the equivalent .NET-specific model and code could be generated. Unfortunately, current tools simply don't have this capability.

## 5.3 MDA-Based Development Tools Are the Next Generation CASE Tool

From a tool perspective, MDA-based development tools are the next generation of Computer Aided Software Engineering (CASE) tools. This is not a bad thing—it is a natural progression of development environments in which modeling, code generation, testing, and deployment capabilities are provided by a single tool. But MDA-based development tools need to go a step further.

The MDA concept adds a level of abstraction to the modeling process. The PIMs represent the desired functionality with no details of the platform. Platform-specific details are added through intense configuration so that code can be generated. We have seen that the difficulty lies in the transformations—mapping of abstract constructs, to platform-specific constructs, to code—which is probably why most tools generate for a single platform. If an organization acquires an MDA-based development tool with the purpose of generating for more than one platform, the focus has to be on these transformations. Creating such transformations is not easy. As we noted earlier, the ability to exchange models between tools and model-to-model transformations is therefore crucial and should be an essential part of MDA-based development tools.

---

[4] It is worth mentioning that UML 2.0 uses Petri-net semantics instead of state machine semantics for activity diagrams.

# 6 Conclusions

MDA is a valid and beneficial approach to software development. What is important is to fully understand the effort involved in using this approach.

Through the model problem process we have partially refuted the hypothesis that the use of MDA reduces development time. Development time can increase greatly for the first application on which it is used due to configuration and transformation modifications. While development time for subsequent applications for the identical platform will reduce significantly, careful analysis has to be done to determine if this reduction is enough to offset that of the first application.

We have refuted the hypothesis that the use of MDA frees the developer from understanding low-level details of the target platform and underlying infrastructure. The use of MDA-based development tools frees the developer from having to write infrastructure code, but it does not free the developer, or person who is doing configuration, from learning the low-level details of the target platform and underlying infrastructure. Ideally, there should be a target architecture expert on the development team.

Current MDA-based development tools present a large initial investment for configuration and potential transformation modification that will only pay off if the tools are going to be used for generating multiple applications for the identical target platform. Changes in the target platform will require re-configuration and modification of the transformations. In the worst case it will require the purchase of an additional tool, which means "starting from scratch" again. Tool capabilities and transformation availability for potential target platforms must be thoroughly investigated before making the investment in an MDA tool.

MDA is a concept that must be fully embraced and implemented by tools before it can become a widespread and cost-effective practice. If not, lack of adequate tooling support could become an adoption barrier to MDA as a technology. Most of the tools that we have investigated are able to generate code for one specific platform from UML models, but the internal representation of these models and transformations is specific to the tool and therefore cannot be shared with other tools. The good news is that OMG continues to make progress in specifications for modeling and transformations. Until tool vendors start fully conforming to these specifications, an MDA-based development tool is nothing more than an automated code generator, and that is not where the real benefit of MDA lies.

Lastly, there must be incentives for tool vendors to go beyond code generation. OMG provides the specification, but somebody has to provide the motivation for vendors to change. In the late 1980s and early 1990s there were several initiatives to develop standard interfaces for CASE tools so that information from one tool could be readily incorporated

into other tools. Even something as simple as configuration management brought on huge disagreements, and vendors were not willing to change their tools for the “common good,” particularly when users expected such integration to come for free. Eventually, these initiatives gave up. If there is no incentive for tool vendors coming from industry, consortia, or user groups to support sharing models and transformations among tools, MDA-based development tools will likely to meet a similar fate as CASE tools.

# References

*URLs are valid as of the publication date of this document.*


**[Apache 05]** The Apache Jakarta Project. *Apache Jakarta Tomcat.* http://jakarta.apache.org/tomcat/ (2005).

**[IO 05]** Interactive Objects. *ArcStyler.* http://www.interactive-objects.com/products/arcstyler_overview.jsp (2005).

**[JBoss 05]** JBoss. *JBoss Application Server.* http://www.jboss.org/products/jbossas (2005).

**[Jython 05]** Jython. *Jython Home Page*. http://www.jython.org/ (2005).

**[Lewis 05]** Lewis, Grace A. & Wrage, Lutz. *Approaches to Constructive Interoperability* (CMU/SEI-2004-TR-020 ESC-TR-2004-020). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005. http://www.sei.cmu.edu/publications/documents /04.reports/04tr020.html.

**[Mellor 02]** Mellor, Stephen & Balcer, Marc. *Executable UML: A Foundation for Model Driven Architecture*. Addison-Wesley Professional, 2002.

**[Middleware 03]** The Middleware Company. *Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach – Productivity Analysis*. June 2003. http://www.omg.org/mda/mda_files/MDA_Comparison-TMC_final.pdf.

**[OMG 02a]** Object Management Group. *XML Metadata Interchange.* http://www.omg.org/technology/documents/formal/xmi.htm (2002).

**[OMG 02b]** Object Management Group. *Meta-Object Facility (MOF).* http://www.omg.org/technology/documents/formal/mof.htm (2002).

**[OMG 03]** Object Management Group. *MDA Guide Version 1.0.1.*, 2003. http://www.omg.org/docs/omg/03-06-01.pdf.

**[OMG 05a]** Object Management Group. *The Data Warehousing, CWM and MOF Resource Page*. http://www.omg.org/technology/cwm/ (2005).

**[OMG 05b]** Object Management Group. *Revised submission for MOF 2.0 Query/View/Transformation RFP (ad/2002-04-10).* March 2005. http://www.omg.org/cgi-bin/doc?ad/05-03-02.

**[Sun 05]** Sun Microsystems. *Java 2 Platform, Enterprise Edition (J2EE).* http://java.sun.com/j2ee/ (2005).

**[Wallnau 01]** Wallnau, Kurt; Hissam, Scott; & Seacord, Robert. *Building Systems from Commercial Components*. New York, NY: Addison-Wesley, 2001.

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*



| | | |
|---|---|---|
| 1. AGENCY USE ONLY<br>(Leave Blank) | 2. REPORT DATE<br>May 2005 | 3. REPORT TYPE AND DATES COVERED<br>Final |
| 4. TITLE AND SUBTITLE<br>Model Problems in Technologies for Constructive Interoperability: Model-Driven Architecture | | 5. FUNDING NUMBERS<br>F19628-00-C-0003 |
| 6. AUTHOR(S)<br>Grace A. Lewis, Lutz Wrage | | |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)<br>Software Engineering Institute<br>Carnegie Mellon University<br>Pittsburgh, PA 15213 | | 8. PERFORMING ORGANIZATION REPORT NUMBER<br>CMU/SEI-2005-TN-022 |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)<br>HQ ESC/XPK<br>5 Eglin Street<br>Hanscom AFB, MA 01731-2116 | | 10. SPONSORING/MONITORING AGENCY REPORT NUMBER |
| 11. SUPPLEMENTARY NOTES | | |
| 12A DISTRIBUTION/AVAILABILITY STATEMENT<br>Unclassified/Unlimited, DTIC, NTIS | | 12B DISTRIBUTION CODE |

13. ABSTRACT (MAXIMUM 200 WORDS)

Model-driven architecture (MDA) is a technology produced and maintained by the Object Management Group (OMG), an open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications. This technical note examines two claims regarding the benefits of MDA, namely, that it (1) reduces development time, and (2) allows the developer to focus on business logic rather than on details about the target platform and architecture. Such advantages would greatly benefit interoperability; as target platforms and underlying infrastructure change, deployment of applications would be quick and easy. This note presents the results of applying the model problem approach to verify these claims.

| | | | |
|---|---|---|---|
| 14. SUBJECT TERMS<br>MDA, Model-Driven Architecture, MDA-based development tool | | | 15. NUMBER OF PAGES<br>37 |
| 16. PRICE CODE | | | |
| 17. SECURITY CLASSIFICATION OF REPORT<br>Unclassified | 18. SECURITY CLASSIFICATION OF THIS PAGE<br>Unclassified | 19. SECURITY CLASSIFICATION OF ABSTRACT<br>Unclassified | 20. LIMITATION OF ABSTRACT<br>UL |